# בינה מלאכותית - תרגיל בית 1

# מגישים: אסף ישראל (041707530), גל כהן (TBD)

## חלק א

1. גרף החיפוש בבעיית הניווט מכיל צמתים (Junctions) כמפגשי רחובות וקשתות המייצגות את בדרכים ביניהם. בתרגום לבעיית החיפוש לכל צומת יש ID שהוא מזהה הצומת ולכל קשת יש משקל שהוא עלות לנסיעה בדרך זו. חישוב משקלה של הקשת תלוי בגורמים שונים התלויים לפי מטרת הבעיה.  
     
   לדוגמא, עבור מציאת הדרך הקצרה ביותר, משקל הקשת יזדקק רק למרחק.  
   עבור מציאת הדרך המהירה ביותר, משקל הקשת יחושב לפי המרחק והמהירות הממוצעת בקשת.  
   ואילו עבור הדרך החסכונית ביותר בדלק נזדקק למרחק, למהירות ולסוג הרכב של המשתמש.  
     
   אנו שומרים סט מינימאלי של פרמטרים על העץ עצמו, ומשתמשים ב-ID של הצומת ובפונקציות ב-CountryMap כדי לחשב ולשלוף את הנתונים הנדרשים לחישוב משקלה של קשת, או לחישוב אלמנטים בהיוריסטיקה.
2. להלן פונקציות המחיר השונות עבור הקשתות בגרף (links):

* עבור Shortest Route, המחיר הוא מרחק הדרך (distance).
* עבור Quickest Route המחיר הוא הזמן הנדרש לעבור קטע דרך כלשהו
* עבור Fuel Saving Route המחיר הוא נצילות הדלק של הרכב בהתחשב בסוג הרכב ומהירות הנסיעה בדרך הנוכחית כפול מרחק הנסיעה.

1. היוריסטיקות קבילות
   1. מרחק אווירי לצומת מטרה. ידוע כי המרחק האווירי הוא המרחק הקצר ביותר בין שתי נקודות כלשהן. לכן בהכרח עבור המסלול האופטימאלי המרחק האווירי קטן או שווה מאורך המסלול P. ולכן היוריסטיקה קבילה.
   2. מרחק אווירי לצומת מטרה מחולק ב120 קמ"ש (120 קמ"ש היא המהירות המקסימלית בגרף). ברור כי כל מסלול אופטימאלי הוא בעל מרחק גדול או שווה למרחק האווירי בין הצומת הנוכחי לסיום. כמו כן בכל קטע דרך המהירות הינה קטנה או שווה ל-120. לכן זמן הנסיעה במסלול האופטימאלי מהצומת הנוכחי לסיום בהכרח קטן או שווה להיוריסטיקה. מכאן שהיא קבילה.
   3. בכל קטע דרך נצילות הדלק גדולה או שווה מהנצילות האופטימאלית עבור סוג רכב כלשהו. מכיוון שהמרחק האווירי בהכרח קטן או שווה מאורך כל דרך מהצומת הנוכחי למטרה, הרי שכמות הדלק במסלול האופטימאלי מהצומת הנוכחי למטרה בהכרח גדולה או שווה לכמות הדלק המחושבת ע"י היוריסטיקה. מכאן שהיא קבילה.

**מה אנחנו צריכים בצומת?**

action is a class that contains the cost of traversing a link

Action:

cost (public field)

State:

getSuccessors():

returns a list of (action,state)

isGoal() (True/False)

איך מחליטים האם הגרף קשיר?

חלק א': 1-3

חלק א': 4

## חלק ב

1. האלגוריתם המוצא על ידנו מנסה למזער את מספר הצעדים הנלקחים בעקבות עדכון קשת כלשהי בגרף, בעת ריצת AStar.  
     
   האלגוריתם מפריד בין שני סוגי עדכונים של קשת:
   1. ירידה במחיר קשת כלשהי – האלגוריתם יחזיר את הצומת העליונה של הקשת חזרה ל-Open, וימשיך בריצה רגילה של AStar.
   2. עליה במחיר קשת – במקרה זה נריץ DFS על תת העץ הנפרס מהצומת העליונה של הקשת עד לחזית החיפוש הנוכחית (כל צמתי Open), שיעדכן את מחירי המסלולים בעקבות השינוי. בהגעה לכל צומת ב-DFS נשאל האם יש לשנות את האב המצביע על הצומת, כיוון שיתכן שכעת יותר כדאי להגיע לצומת חיפוש כלשהו ממסלול אחר דווקא. לשם כך, נחזיק מיפוי בין כל מצב (state) לבין כל צמתי החיפוש (Node) הידועים עבורו (בין אם הם ב-Open או ב-Closed). עבור כל צומת חדש ש-DFS מגיע אליו, נשאל מיהו צומת החיפוש (תזכורת: צומת חיפוש = מצב + צומת החיפוש האב + מחיר המסלול מהשורש עד אליו) בעל המשקל הכדאי ביותר. בפעולה זו נזכור לא לשקול את צומת החיפוש שמסלולו נפסל בעקבות העדכון.  
      לאחר שה-DFS יסיים, נמשיך בריצה רגילה של ה-AStar.
2. הוכחת נכונות (קבילות)  
   נפריד למקרים לפי סוג עדכון הקשת:
   1. נשים לב שירידה במחיר קשת אינה יכולה לשנות מסלול בין צומת ב-Closed לבין צומת ב-Open, שכן המסלול גם לפני השינוי היה הקל ביותר, וכעת הוא נעשה קל אפילו יותר. מכאן שמרגע המעבר של צמתי הקשת ל-Open מובטח שצמתים אלו יהיו ראשונים בתור ה-Open (כיוון שתת המסלול מהשורש אליהם, קל יותר מכל מסלול מהשורש לחזית הקודמת) ולכן AStar יעדכן את עצמו במהלך ריצתו.
   2. עבור מקרים בהם משקל הקשת עלה התמונה יותר מסובכת, כיוון שכעת כל תת העץ הנפרס מהצמתים שעודכנו עד החזית כבר לא מעודכן, ומסלולים אל צמתי החיפוש בו יכולים להשתנות. לשם כך נריץ את ה-DFS עד החזית. נשים לב שה-DFS יגיע אל כל צומת שמושפע בצורה זו או אחרת מהשינוי במשקל הקשת ויעדכן מסלולים במקרה הצורך. העדכון יסתיים ברגע שהגענו אל החזית הנוכחית של AStar ועדכנו את מחירי המסלולים בה (למעשה החזרנו תור Open חדש).  
      מרגע זה AStar יכול להמשיך כרגיל, והנכונות נובעת מכך שכל צומת ב-Open מייצג מסלול חיפוש זול ביותר מהשורש אליו.
3. ניתוח ביצועי האלגוריתם:  
   השווינו את ביצועי האלגוריתם כנגד אלגוריתם ה-Baseline המתחיל את ריצתו מהתחלה בכל פעם שמתבצע עדכון.  
   ביצענו שתי השוואות של האלגוריתמים, כאשר בכל אחת נבחנו 25 זוגות מסלולים זהים. כיוון שאין דרך אמיתית להגריל את אותם קשתות עדכון בכל ריצה של שתי האלגוריתמים (כיוון שהם תלויות בצמתים שפותחו בכל ריצה) אנו משווים את הטרנד של כל הריצות מתוך תקווה שאותו מספר ממוצע של עדכונים פותח בשני האלגוריתמים.
   1. ההרצה הראשונה איפשרה מספר כלשהו של עדכונים מפונקצית GetSpeedUpdates. פה איבחנו יתרון קל ל-Baseline במספר פיתוחי הצמתים. אנו מייחסים זאת לעובדה שעדכון המכיל מספר צמתים גורם להתחלה חדשה אחת של Baseline, בעוד האלגוריתם שלנו מפתח תת עץ שלם עבור כל עדכון (עבור עדכון חיובי יפותח התת עץ במסגרת ה-DFS, ואילו עבור עדכון שלילי הוא יעודכן במסגרת המשך ריצת ה-AStar).  
        
      גרף 1 מתאר את ההפרש במספר פיתוחי הצמתים בין Baseline לבין האלגוריתם שלנו Complex. הפרש שלילי מצביע על יתרון ל-Baseline במספר פיתוחי הצמתים (פיתח פחות צמתים). כפי שניתן לראות הממוצע מצביע על יתרון ל-Baseline.

Figure 1הפרש פיתוחי הצמתים בין Baseline ו-Complex

גרף 1 הפרש פיתוחי הצמתים בין Baseline ו-Complex, מספר קשתות בעדכון

* 1. בניסיון להוכיח את הטענה בה Baseline מנצל איתחול אחד עבור מספר עדכונים, הרצנו ניסוי נוסף בו מספר העדכונים שהוחזר מ-GetSpeedUpdates בכל הפעלה היה אחד בלבד.  
       
     כפי שניתן לראות מגרף 2 כאן כבר ישנו יתרון קל ל-Complex, שכן כל עדכון מפתח מחדש רק תת-עץ בגרף ולא את כל העץ מהתחלה.

גרף 2הפרש פיתוחי הצמתים בין Baseline ו-Complex, קשת אחת לעדכון

נציין רק שההפרשים הנ"ל הם לרוב זעומים יחסית למספר הצמתים הכולל שפותח בכל אלגוריתם, ולרוב מסתכמים בכ-0.05-0.01% מכלל הצמתים שמפותחים. הסיבה לכך היא ההגבלה על מספר הקשתות שמתעדכנות (10) הינה קטנה במיוחד ביחס לאורך המסלול הממוצע. כנראה שעבור מספר הגרלות רב יותר הטרנד יהיה יותר ברור לטובת Complex.